基于(n,r,k) Fork-Join 队列分析的 NWR 数据库写延时模型 *

王华进 [1,2]，黎建辉 [1†]，沈志宏 [1]

(1. 中国科学院计算机网络信息中心，北京 100190; 2. 中国科学院大学，北京 100049)

摘　要：NWR 数据库的写延时估计，可用于发现实现集群构建和运行成本最小化的节点数量、副本因子的配置组合。现有基于基准测试或模拟队列的方法受限于特定的测试配置和测试环境，只能给出写延时随配置变动的粗略结果。从分析 NWR 数据库 Cassandra 的写操作的（n，r，k）Fork-Join 队列结构入手，给出了该类队列期望逗留时间的解析解和 NWR 数据库写延时的理论模型，可用于建立更完备的写延时结论。分别在模拟队列和 Cassandra 集群上验证了（n，r，k）队列解析解和写延时模型的准确性。

关键词：数据库队列；副本因子；一致性；写延时

中图分类号：TP311　　doi: 10.3969/j.issn.1001-3695.2017.09.0878

Write latency model for NWR databases based on (*n*, *r*, *k*) Fork-Join queueing analysis

Wang Huajin[1, 2], Li Jianhui[1†], Shen Zhihong[1]

(1. *Computer Network Information Center*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*; 2. *University of Chinese Academy of Sciences*, *Beijing* 100049, *China*)

Abstract: The exact approximation of write latency for NWR databases under various consistency levels can serve the building and operating of database clusters, by finding the optimal combination of cluster size and replication factor that minimizes the building and operating cost. Existing benchmarking or queue simulating based approaches can only give incomplete results as they are limited to specific configurations and testbeds. This paper depicted the first close-form analysis of (n, r, k) fork-join queueing process of Cassandra (a typical NWR database) write operations, based on which this paper proposed the first theoretical write latency model for NWR databases. The model is capable of giving more comprehensive latency results. Experiments validated the close-form analysis of (n, r, k) fork-join queues and the write latency model respectively on simulated queues and a Cassandra cluster.

Key Words: database queue; replication factor; consistency level; write latency

0　引言

NWR 是一种流行的 Key-Value 数据库一致性策略：1）每个 Key-Value 对被存储在位于 N 个位于不同节点的副本中；2）完成读操作至少需要读取 R 个副本中最新的数据；3）完成写操作至少需要成功写入 W 个副本；4）当 $R+W>N$ 时可以保证读到最新写入的数据。Amazon Dynamo[1] 和 Apache Cassandra[2]是两个流行的采用了 NWR 一致性策略的 Key-Value 数据库（以下简称 NWR 数据库）。

在 NWR 数据库中，写操作是通过 Fork-Join 队列分发到各个副本上并发执行的：a）客户端提交的写操作在控制节点被复制（Fork）为 N 份，分发到目标记录所在的 N 个副本节点上；b）每个副本节点写完成后，给出控制节点操作完成的答复；c）控制节点接收（Join）到第 W 个答复后，给出客户端写请求已成功执行的响应。

显然在 NWR 数据库中，写操作的延时由集群节点数量 L、副本因子 N、一致性水平 W 共同决定。一般而言，L 越大、N 越大、W 越小，延时越小。但 L 越大，集群的构建成本越大；N 越大，集群在存储和操作上的运行成本越大。鉴于 NWR 数据库集群往往针对特定目标场景负载的 SLA（服务水平协议）构建，需满足 SLA 中对各种一致性水平下的写延时的要求。通行做法是，提供尽可能大规模的集群和较高设置的副本因子，以保证在极端负载压力下仍能满足写延时要求，但这往往导致系统资源和能源的浪费。准确估计各种一致性水平和负载压力下的 NWR 数据库的写延时，可用于发现满足延时要求的最小的 L、N 组合，从而为成本最小化集群的构建和配置提供重要

基金项目：国家“973”计划资助项目（2016YFB1000600，2016YFB0501900）

作者简介：王华进（1987-），男，山东茌平人，博士研究生，主要研究方向为分布式系统；黎建辉（1973-），男（通信作者），北京人，正研级高工，博士，主要研究方向为科学大数据管理、数据出版、数据挖掘（lijh@cnic.cn）；沈志宏（1977-），男，北京人，正高级工程师，博士，主要研究方向为科学大数据管理．

参考依据。

现有针对 Cassandra 等 NWR 数据库的操作延时分析方法主要有：a）基于基准测试的方法[3-6]，采用分布式数据库的主流基准测试工具得出实际集群在各种节点数量、副本因子、一致性水平、负载压力下的读写操作延时。但限于基准测试不能穷尽所有的配置组合，这些工作只是各自给出了延时随部分配置组合变动的粗略趋势，并不能证明其结论具有普遍适用性（完备性）;b）基于模拟队列的方法[7-13]，构建与 Cassandra 等数据库的读写操作执行过程等价的模拟队列网络（Queue Network），通过运行模拟队列而非实际集群获取延时数据，以减轻测试的开销。由于模拟队列的结构和参数都只是对真实集群在特定配置和负载下的近似，其测试结果的可靠性要差于基准测试，并且所得出的结论同样是粗略、不完备的。

本文给出了 NWR 数据库的写延时理论模型。该模型的输入参数为最高一致性水平（ $W=N$ ）下的写操作在不同配置组合下的期望延时，输出为各种较弱一致性水平（ $W=1,2,\ldots,N-1$ ）下的写操作在相关配置组合下的期望延时的估计值。输入参数既可以通过理论计算获取，也可以借助基准测试获取。借助该模型进行同等规模的基准测试可以获取更为详细和完备的写延时结论。该理论模型的推导建立在对 Cassandra 写过程的排队结构 (n,r,k) Fork-Join 队列的逗留时间的定量分析上。现有 (n,r,k) 队列逗留时间的研究仅给出了较为粗略的上下界[14]。本文通过证明 (n,r,k) 队列的期望逗留时间为基本 Fork-Join 队列 (i,i,i) 队列（ $1\le i\le r$ ）的期望逗留时间的线性组合，将较为困难的 (n,r,k) 队列分析转换为较为容易的 (i,i,i) 队列分析。最后，通过解剖 Cassandra 写过程的排队结构，将 (n,r,k) 队列的定量分析应用于实际数据库写延时的估计上。

本文的主要贡献为：a）第一次给出了 (n,r,k) Fork-Join 队列期望逗留时间的确切解析表达式，并在独立同指数服务时间分布的 (n,r,k) Fork-Join 模拟队列上验证了该表达式的准确性；b）第一次给出了 NWR 数据库写延时的理论模型，并在实际 Cassandra 集群上验证了该理论模型的适用性。

1　非清除 (n, r, k) Fork-Join 队列分析

本文考虑一个由 n 个节点构成的集群，其中每个节点 i （ $1\le i\le n$ ）处理同类作业子任务时的服务时间 X_i 具有相同的概率分布（即节点同质）。每个节点上有一个先到先服务的子队列 q_i，假定子队列容纳能力无限。令：到达率 λ 表示单位时间内提交到集群的作业数量，服务率 μ 表示单位时间内一个子队列可以完成服务的子任务数量，负载因子 $\rho=\dfrac{\lambda}{\mu}$ 。

定义 1　非清除 (n,r,k) Fork-Join 队列。如图 1 所示，在一个由 n 个子队列组成的非清除 (n,r,k) Fork-Join 队列中：1）客户端提交的作业在到达控制节点时被复制（Fork）为 r 个相同开销的子任务，每个子任务被分配到 r 个不同节点上运行，每个节点被选中运行子任务的概率相同；2）节点执行完子任务后向控制节点发回答复；3）控制节点收到（Join）前 k 个节点的答复后，对客户端发出作业完成的响应；4）剩余 $r-k$ 个节点上的子任务会继续执行（非清除），但其结果无须控制节点处理。

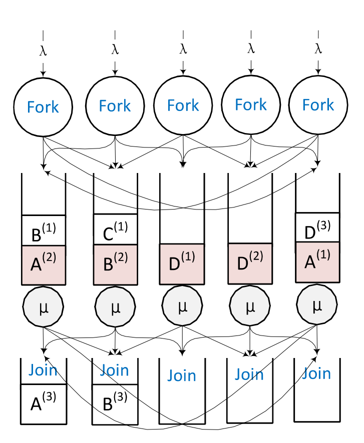


图 1　一个非清除 $(5,3,2)$ Fork-Join 队列示例：红色子任务正在被服务

定义 2　期望逗留时间 $\mathbb{T}^{\rho}_{n,r,k}$ 。非清除 (n,r,k) Fork-Join 队列的期望逗留时间 $\mathbb{T}^{\rho}_{n,r,k}$ 为：当该队列在负载因子 ρ 下达到稳态时，一个新作业从在控制节点被 Fork 开始计时到在控制节点完成 Join 结束计时，所经历的时间间隔的期望值。

1.1　一般服务时间队列的分析

引理　设有 2 个服务时间分布相同的 Fork-Join 队列，其中一个为处于负载因子 ρ 下的非清除 (n,r,k) 队列，另一个为处于负载因子 $\dfrac{r\rho}{n}$ 下的非清除 (r,r,k) 队列。在忽略了渐进独立性[15]的情况下，上述两个队列的期望逗留时间相同，即：

$\mathbb{T}^{\rho}_{n,r,k}=\mathbb{T}^{\frac{r\rho}{n}}_{r,r,k}$ 。

证明　在上述 (n,r,k) 队列和 (r,r,k) 队列中，一个新到达的作业都会有 r 个子任务分配到 r 个不同子队列中，并等待前 k 个子任务完成。由于每个节点被选中运行子任务的概率相同，(n,r,k) 队列的每个子队列承受的子任务到达率与 (r,r,k) 的子队列承受的子任务到达率均为 $\dfrac{r\lambda}{n}$ ，又由于所有子队列的服务时间分布相同，因此在两种队列达到稳态后，所有子队列的期望长度相同。设随机变量 S_i 表示一个子任务在子队列 q_i 中的逗留时间，次序统计量 $S_{(r,k)}$ 表示 $S_1,S_2,\ldots,S_r$ 的第 k 次序统计量。在忽略了渐进独立性[15]的情况下： $\mathbb{T}^{\rho}_{n,r,k}=E[S_{(r,k)}]$ ， $\mathbb{T}^{\frac{r\rho}{n}}_{r,r,k}=E[S_{(r,k)}]$ 。

定理 1　非清除 (n,r,k) Fork-Join 队列的期望逗留时间为

$$\mathbb{T}^{\rho}_{n,r,k}=\sum_{i=k}^{r}W_i^{r,k}\mathbb{T}^{\frac{r\rho}{n}}_{i,i,i}$$

其中：常量系数 $W_i^{r,k}=\sum_{j=k}^{i}\binom{r}{j}A_i^{r,j}$ ，

$$A_i^{r,j}=\begin{cases}1 & i=j,\\ -\sum_{l=1}^{i-j}\binom{r-i+l}{l}A_{i-l}^{r,j} & j+1\le i\le r.\end{cases}$$

证明　根据引理，$\mathbb{T}^{\rho}_{n,r,k}=\mathbb{T}^{\frac{r\rho}{n}}_{r,r,k}$。根据文献[16]的定理 5 可知：

$$\mathbb{T}^{\frac{r\rho}{n}}_{r,r,k}=\sum_{i=k}^{r}W_i^{r,k}\mathbb{T}^{\frac{r\rho}{n}}_{i,i,i}。$$

值得注意的是：定理 1 的成立无须假设子队列的服务时间相互独立。

定理 2　非清除 (n,r,k) Fork-Join 队列的期望逗留时间 $\mathbb{T}^{\rho}_{n,r,k}$ 的上下界为：$\mathbb{T}^{\frac{r\rho}{n}}_{k,k,k}\ge\mathbb{T}^{\rho}_{n,r,k}\ge E[X_{(r,k)}]+\frac{r\lambda E[X^2_{(r,1)}]}{2(n-r\lambda E[X_{(r,1)}])}$；或 $\mathbb{T}^{\frac{r\rho}{n}}_{k,k,k}\ge\mathbb{T}^{\rho}_{n,r,k}\ge E[X_{(r,k)}]+\frac{r\lambda E[X^2_{(r,k)}]}{2(n-r\lambda E[X_{(r,k)}])}$，如果服务时间是独立同指数分布的。其中 $E[X_{(r,k)}]$ 表示 r 个子队列的服务时间变量 $X_1,X_2,\ldots,X_r$ 的第 k 次序统计量的期望。

证明　$\mathbb{T}^{\rho}_{n,r,k}\equiv\mathbb{T}^{\frac{r\rho}{n}}_{r,r,k}\equiv E[S_{(n,k)}]\le E[S_{(k,k)}]\equiv\mathbb{T}^{\frac{r\rho}{n}}_{k,k,k}$ 是显然的。

根据文献[14]的定理 4 可推得到达率 $\frac{r\lambda}{n}$ 下的清除型 (r,r,k) Fork-Join 队列（即作业 join 完成后，剩余任务被从各子队列和运行态中立即清除）的期望逗留时间下界为 $E[X_{(r,k)}]+\frac{r\lambda E[X^2_{(r,1)}]}{2(n-r\lambda E[X_{(r,1)}])}$，又根据文献[16]的定理 10，独立同指数服务时间分布的上述清除型队列存在一个更紧密的下界 $E[X_{(r,k)}]+\frac{r\lambda E[X^2_{(r,k)}]}{2(n-r\lambda E[X_{(r,k)}])}$，显然此处的非清除 (r,r,k) 队列的期望逗留时间的下界不小于上述下界。

根据定理 1 和定理 2，非清除 (n,r,k) 队列的期望逗留时间 $\mathbb{T}^{\rho}_{n,r,k}$ 的求解可以转换为较容易计算的 (i,i,i) 队列的期望逗留时间 $\mathbb{T}^{\rho}_{i,i,i}$ 和子队列服务时间的次序统计量 $X_{(r,k)}$。

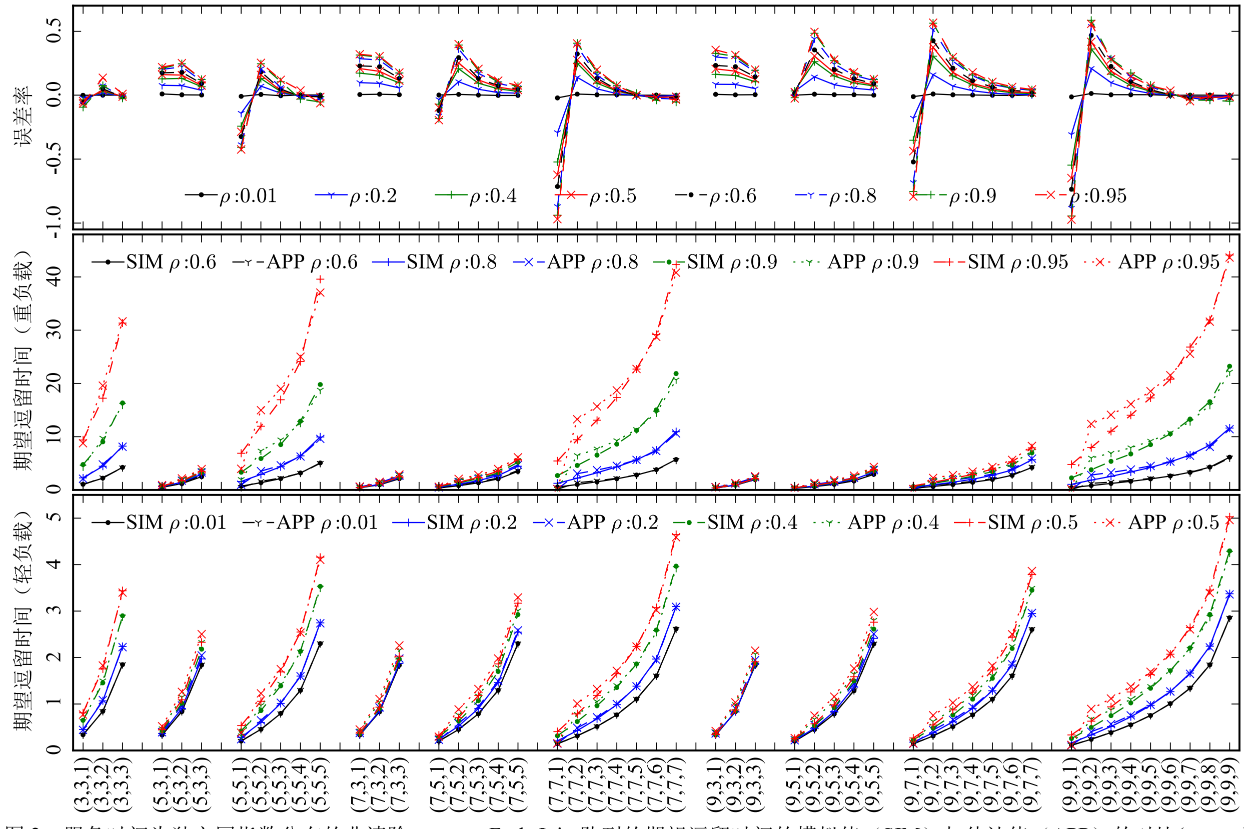


图 2　服务时间为独立同指数分布的非清除 (n,r,k) Fork-Join 队列的期望逗留时间的模拟值（SIM）与估计值（APP）的对比($\mu=1$)

1.2　指数服务时间队列的估计

鉴于独立同指数服务时间分布的 (i,i,i) Fork-Join 队列的期望逗留时间 $\mathbb{T}^{\rho}_{i,i,i}$ 存在较为准确的 Nelson 估计方法[17]，本文在定理 3 中给出该类队列期望逗留时间的估计值。

定理 3　服务时间为独立同指数分布的非清除 (n,r,k) Fork-Join 队列的期望逗留时间:

$$T^{\rho}_{n,r,k}\simeq\min\left(\max\left(NelsonT^{\rho}_{n,r,k},LowT^{\rho}_{n,r,k}\right),UpT^{\rho}_{n,r,k}\right)$$

其中：

$$NelsonT^{\rho}_{n,r,k}=\begin{cases}\frac{nr}{\mu(n-r\rho)}+\frac{12n-r\rho}{88\mu(n-r\rho)}\times\sum_{i=2}^{r}W_i^{r,1}\left[\frac{11nH_i+4r\rho(H_2-H_i)}{nH_2}\right] & k=1,\\ \frac{12n-r\rho}{88\mu(n-r\rho)}\times\sum_{i=k}^{r}W_i^{r,k}\left[\frac{11nH_i+4r\rho(H_2-H_i)}{nH_2}\right] & k\ge 2.\end{cases}$$

$$LowT^{\rho}_{n,r,k}=\frac{H_r-H_{r-k}}{\mu}+\frac{r\rho(H_{r(r-r\rho/n)}-H_{(r-k)(r-k-r\rho/n)})}{n\mu}$$

$$UpT^{\rho}_{n,r,k}=\begin{cases}\dfrac{n}{\mu(n-r\rho)} & k=1,\\[2ex] \dfrac{12n-r\rho}{8\mu(n-r\rho)}\left[\dfrac{H_k}{H_2}+\dfrac{4r}{11n}\left(1-\dfrac{H_k}{H_2}\right)\rho\right] & k\geq 2.\end{cases}$$

上述公式中，$H_m=\sum_{i=1}^{m}\frac{1}{i}$ 为调和级数，$H_{m(m-\rho)}=\sum_{i=1}^{m}\frac{1}{i(i-\rho)}$ 。

证明 将文献[17]的式(6)代入定理 1 可以得出 $T^{\rho}_{n,r,k}$ 的具体估计值 $NelsonT^{\rho}_{n,r,k}$；将文献[17]的式(6)代入定理 2 的下界可以得出 $T^{\rho}_{n,r,k}$ 的下界 $LowT^{\rho}_{n,r,k}$。很容易得出 r 个独立同指数分布的服务时间变量的第 k 次序统计量的期望 $E[X_{(r,k)}]=\frac{H_r-H_{r-k}}{\mu}$，将该值代入定理 2 的上界得出 $UpT^{\rho}_{n,r,k}$。由于单独使用 $NelsonT_{n,r,k}$ 作为估计值可能带来较大的误差[16]，采用上下界 $UpT^{\rho}_{n,r,k}$、$LowT^{\rho}_{n,r,k}$ 对估计值进行了限定。

如图 2 所示，采用模拟值（SIM）对从定理 3 中得出的估计值（APP）进行了验证。针对每一组 (n,r,k)，给模拟器 Fokulator-p[1]施加指定负载压力（ρ）的作业到达流。当队列到达稳态后，以 10%的采样率获得 100000 个样本作业的延时均值为模拟值。误差率的计算公式为 $\frac{\text{APP}}{\text{SIM}}-1$。

从图 2 中可以看出，与模拟值对比，估计值的误差在大部分情况下较为可控。只有当负载特别重($\rho>0.8$)且 $k<\frac{2}{r}$ 时误差较为明显（大于 25%）。这是由于：当 ρ 较大时，通过 Nelson 估计方法得出的 $T^{\frac{r\rho}{n}}_{i,i,i}$ 的误差较大；而 k 较小时，更多的 $T^{\frac{r\rho}{n}}_{i,i,i}$ 误差项被积累到了估计值里。未来本文可以通过改进指数队列 $\mathbb{T}^{\rho}_{i,i,i}$ 的估计方法，降低该误差。

2　Cassandra 写操作排队分析

Cassandra 写操作的执行架构如图 3 所示。

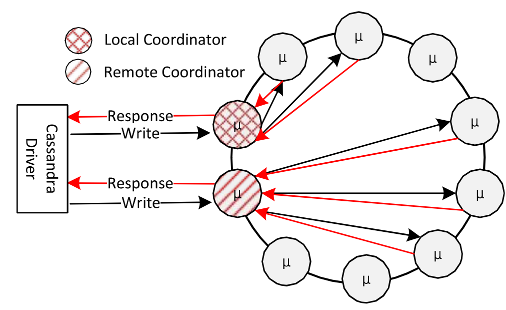


图 3　Cassandra 写操作的执行架构

a）客户端（Cassandra Driver）向由 n 个节点构成的 Cassandra 集群提交写操作请求，所连接的节点成为 Coordinator；b）Coordinator 将写命令发送到存储有目标记录副本的 r 个节点上（即副本因子为 r）；c）当 Coordinator 从副本节点收到的回复数量达到一致性要求的下限 k 时，就给出客户端写完成的响应。当 Coordinator 本身是副本节点时，称之为本地 Coordinator，否则为远程 Coordinator。默认配置下，一条 Key-Value 记录顺时针连续地存放在集群的 r 个节点上，其中第 1 个节点的位置是通过对 Key 进行散列确定的。这种存储划分方式可以基本保证各个节点存储同等数量的 Key-Value 记录。

Cassandra 内部采用了 SEDA (Staged Event-Driven Architecture)[18]的设计框架，将写操作表达为一个由 Native-Transport-Requests、Mutation、RequestResponse 等步骤组成的串行操作序列。每个步骤包含一个处理线程池，空闲线程从一个事件到达队列中取操作请求并执行，执行结果送入下一个步骤的到达队列。写操作的具体实现为：a）客户端的请求送到 Coordinator 的 Native-Transport-Requests 阶段的到达队列，该阶段首先获取目标副本所在的节点信息，然后通过 MessagingService 组件将写操作传输到每个副本节点的 Incoming 队列；b）副本节点上的 Mutation 阶段从 Incoming 队列中获取并执行写请求（追加 Commitlog 和写入 Memtable）；c）副本节点通过 MessagingService 组件将写操作的执行结果传输回 Coordinator 的 Incoming 队列；d）Coordinator 的 RequestResponse 阶段读取并计算 Incoming 队列中答复写操作的消息数量；e）若该数量达到一致性水平要求的下限，则向客户端发出最终响应。由于上述各阶段的队列长度上限为 $2^{31}-1$，因此当不对操作施加 timeout 限制时，所有阶段的到达队列都可近似为非损失制。

如上所述，Cassandra 写操作的执行过程非常适合采用本文定义的非清除 (n,r,k) Fork-Join 队列进行描述。本文将写操作的每个内部执行步骤视为队列服务过程的一部分，得出如图 4 所示的简化 Cassandra 写操作排队模型。

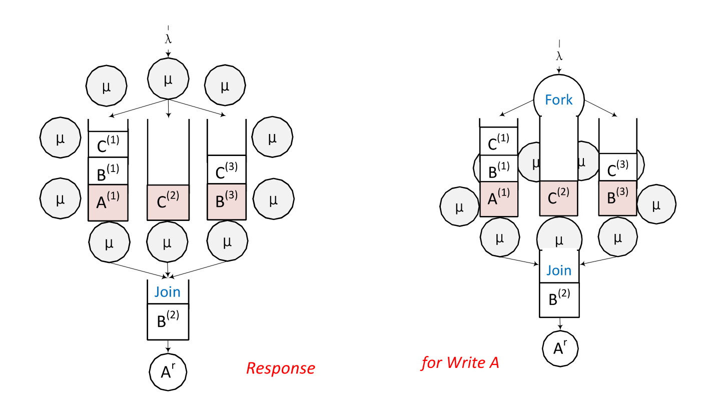


图 4　Cassandra 写操作的简化 Fork-Join 排队模型：远程写（左）、本地写（右）

Cassandra 客户端可以设置不同的负载均衡策略来控制 Coordinator 的选取：TokenAwarePolicy 优先使用本地 Coordinator，若所有本地 Coordinator 过载，则尝试使用远程 Coordinator；RoundRobinPolicy 等概率选取集群中的所有节点

[1] https://github.com/excelwang/forkulator-p

作为 Coordinator；HostFilterPolicy 可以指定具体节点做 Coordinator。由于本地 Coordinator 下的写操作（本地写）的服务时间远小于远程 Coordinator 下的写操作（远程写）的服务时间，写延时模型必须反映本地写的比例。据此本文得出如下写延时模型。

定理 4　运行在一个同质集群上的 Cassandra 数据库的写操作期望延时模型为

$$\mathbb{T}^{\rho,l}_{n,r,k} = l\times LT^{\rho}_{n,r,k} + (1-l)\times RT^{\rho}_{n,r,k}$$

其中：

$$RT^{\rho}_{n,r,k} = \sum_{i=k}^{r} W_i^{r,k} RT^{\frac{r\rho}{n}}_{i,i,i}$$

$$LT^{\rho}_{n,r,k} = \begin{cases} LT^{\frac{r\rho}{n}}_{1,1,1} & k=1, \\ \sum_{i=k-1}^{r-1} W_i^{r-1,k-1} RT^{\frac{r\rho}{n}}_{i,i,i} & k\ge 2. \end{cases}$$

模型输入参数：a）l 为本地写的比例；b）$LT^{\frac{r\rho}{n}}_{1,1,1}$ 为本地写在负载压力为 $\frac{r\rho}{n}$，节点数、副本因子、一致性水平均为 1 时的期望延时；c）$RT^{\frac{r\rho}{n}}_{i,i,i}$，$(k-1\le i\le r)$ 为远程写在负载压力为 $\frac{r\rho}{n}$，节点数（不计入 Coordinator 节点）、副本因子、一致性水平均为 i 时的期望延时。上述参数共计 r-k+4 个。

参数 l 可以从负载均衡策略中推出，如 RoundRobinPolicy 对应的 $l=\frac{r}{n}$、轻量级负载下 TokenAwarePolicy 对应的 $l=1$；参数 $LT^{\frac{r\rho}{n}}_{1,1,1}$ 和 $RT^{\frac{r\rho}{n}}_{i,i,i}$ 既可以从本地写/远程写的服务时间的概率分布中估算得出（参见节 1.2，及一般服务时间分布的 (i,i,i) 队列的期望逗留时间的估计方法[19,20]），也可以用基准测试测得的延时样本均值估计。

3　实验

本文通过实验测得由 6 个同质节点组成的 Cassandra 集群在不同副本因子（r）、一致性水平（k）、负载均衡策略（l）、负载压力（ρ）下的写延时，并与基于定理 4 的延时模型得出的估计值进行比较，以验证延时模型的适用性。其中：a）测出值通过 Cassandra 自带的查询追踪（Trace）功能得出，其精度为微秒级（$10^{-6}s$）。由于查询追踪本身的开销不可忽略，限定一个写操作被追踪的概率不大于 6%。测试中，每秒提交写操作的数量标记为 λ，写操作的提交间隔依照指数分布 $exp(\lambda)$ 选取;b）预测值通过将从集群测出的 $LT^{\frac{r\rho}{n}}_{1,1,1}$ 和 $RT^{\frac{r\rho}{n}}_{i,i,i}$ 等参数值代入定理 4 得出。

为得出可比较和可重复的测出值，本文尽可能固定除副本因子、一致性水平、负载压力、负载均衡策略之外的变量（如写操作的数据量），并采用 6 台位于同一机架的相同配置的物理机节点作为测试床。每个测试节点的配置为：双路 Intel Xeon E5-2603 v3 处理器、64GB 内存、3.6 TB 机械磁盘、万兆以太网互连。测试用 Cassandra 版本为 3.11.0。副本因子在 1~5 变动，一致性水平从 ONE ($k=1$)、QUORUM ($k=\left\lceil\frac{n+1}{2}\right\rceil$)、ALL ($k=n$)中选取，施加的目标负载压力标记为 λ。本文为各副本因子预先创建一个表空间（keyspace），并各存入 1 亿条 Key-Value 记录。

为获取定理 4 给出的延时模型的输入参数并详细界定该延时模型的适用范围，本文设计了“微测试”和“扩展测试”两种实验方案。

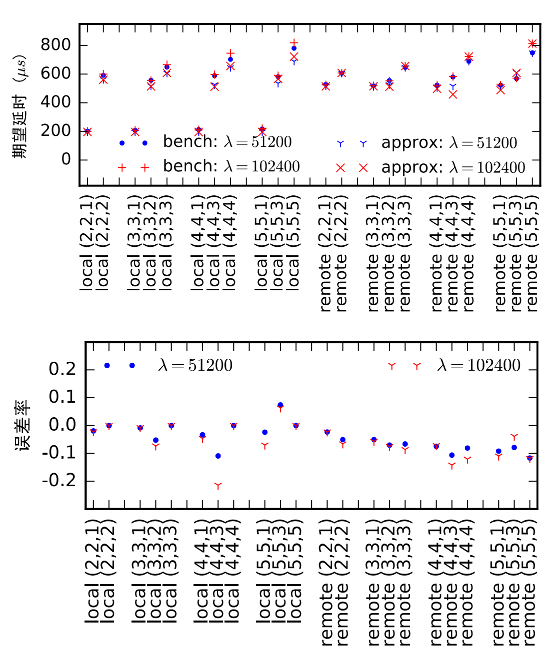


图 5　估计值与微测试测出值的对比

（local：全部本地写，remote：全部远程写）

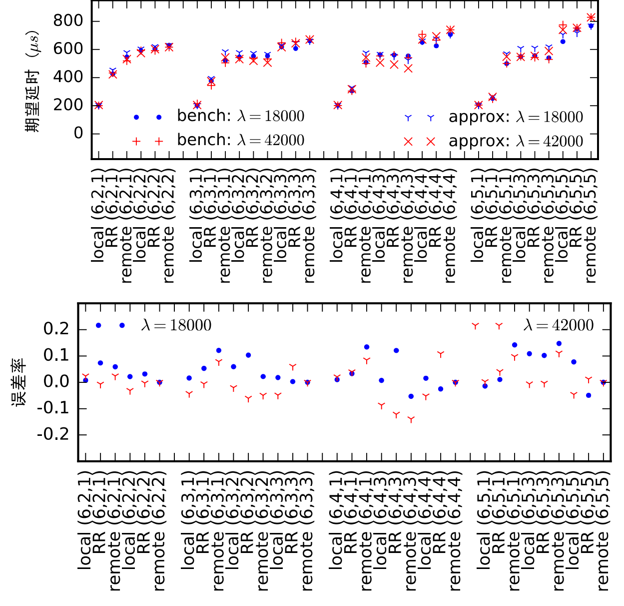


图 6　估计值与扩展测试测出值的对比（local：全部本地写，remote：全部远程写，RR: RoundRobinPolicy）

3.1 微测试

微测试在不同副本因子、一致性水平、负载压力、负载均衡策略下大量重复执行针对同一个 Key-Value 记录的写操作，以：1）获取延时模型的输入参数 $LT_{1,1,1}^{\frac{r\rho}{n}}$ 和 $RT_{i,i,i}^{\frac{r\rho}{n}}$；2）校验基于 (n,r,k) Fork-Join 队列的简单形式 (r,r,k) 队列的延时模型估计 Cassandra 数据库单一写操作延时的适用性。其中，负载均衡策略是基于 HostFilterPolicy 定制实现的全部本地写策略（local）和全部远程写策略（remote）。这两种策略对应的延时模型 l 参数值分别为 1 和 0。

每种配置组合的测试结果抽样率为 1%，样本数为 60000。实验结果如图 5 所示。延时模型的估计值（approx）较为接近测出值（bench），基本验证了延时模型估计单一写操作延时的适用性，其中误差率的计算为 $\frac{\text{approx}}{\text{bench}}-1$。

注意图中的 λ 只是测试程序施加的压力目标，受限于 Python 测试脚本自身的开销、基于 sleep()函数间隔相邻操作的精度、Cassandra 集群的实际处理能力，实际负载压力要低于目标压力。

3.2 扩展测试

扩展测试在不同副本因子、一致性水平、负载压力、负载均衡策略下大量执行混合在一起的针对不同 Key-Value 记录的写操作，以：a）获取延时模型的输入参数 $LT_{1,1,1}^{\frac{r\rho}{n}}$ 和 $RT_{i,i,i}^{\frac{r\rho}{n}}$；b）校验 Cassandra 写延时模型估计混合写操作延时的适用性。混合测试负载的构造确保了每个节点等概率地成为 Coordinator，且承受同等压力的副本写入请求，以尽可能反映实际场景下负载在各节点上的均衡分配特性。其中，被测试的负载均衡策略包括全部本地写、全部远程写和 RoundRobinPolicy，这 3 种策略对应的延时模型 l 参数值分别为 1、0、r/n。

每种配置组合的测试结果抽样率为 6%，全部本地写、全部远程写、RoundRobinPolicy 的样本数分别为 4200、60000、4200。实验结果如图 6 所示：延时模型的大部分估计值（approx）非常接近测出值（bench），基本验证了延时模型估计混合写操作延时的适用性。注意图中的 λ 同样为目标值，实际达成的负载压力水平低于该值。

值得注意的是，图 6 中的基准测试测出值和延时模型估计值均显示了：在相同副本因子、一致性水平、负载压力下，全部本地写、RoundRobinPolicy、全部远程写这 3 种负载均衡策略的写延时呈递增趋势。从该结果可以推断：TokenAwarePolicy 均衡策略（该策略与全部本地写策略相近，l 值均为 1）要优于 RoundRobinPolicy 均衡策略。

4 讨论

如引言所述，该理论模型的输入参数为最高一致性水平（$W=N$）下的写操作在不同配置组合下的期望延时，输出为各种较弱一致性水平（$W=1,2,\ldots,N-1$）下的写操作在相关配置组合下的期望延时的估计值。如节 3 所述，本文采用基准测试测出理论延时模型的输入参数。显然，在进行同等开销的基准测试时，理论延时模型可以得出比仅采用基准测试多 N-1 倍的延时结论。

节 3 初步验证了本文提出的 NWR 数据库理论延时模型的准确性，但通过基准测试获取模型参数的开销较大。更为理想的方案是，只需通过实验测出无压力下本地写和远程写操作的服务时间分布，然后借助该分布计算出上述参数在各种负载压力下的估计值。

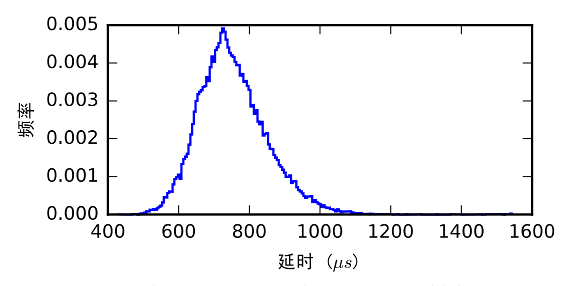


图 7　Cassandra 集群的远程写服务时间分布（样本量 50000）

目前独立同指数服务时间分布的 (i,i,i) Fork-Join 队列的期望逗留时间存在较为准确的估算方法[17]。但本文实验测出的服务时间分布并不符合指数分布（见图 7）。尽管对于一般服务时间分布的 (i,i,i) 队列，存在一些基于插值[19]和回归[20]的估计方法，但其准确性依赖于所测出的服务时间分布的精度。受限于测量精度，本文测出的服务时间的分布并不十分稳定。这也是目前 Fork-Join 队列的理论研究（如文献[14,21]）鲜有能被实际集群验证的原因，大部分工作只是给出了在模拟队列上的验证。未来，本文将进一步研究更为鲁棒的一般服务时间 (i,i,i) 队列的估计方法。

5 相关工作

5.1 Fork-Join 队列分析

虽然 (i,i,i) 队列是最基本的 Fork-Join 队列，当 $i>2$ 时其期望逗留时间仍然没有确切的解析解，只有一些估计方法存在。如：Nelson 等人[17]给出了指数服务时间分布的 (i,i,i) 队列的期望逗留时间估计式；针对服务时间分布较为一般的 (i,i,i) 队列，Varma 等人[19]提出了基于插值的估计方法，Thomasian 等人[20]提出了基于测试结果回归分析的的估计方法，Rizk 等人[22]给出了基于鞅(martingales)的上界，Fidler 等人[23]给出了多步骤 Fork-Join 队列网络的上下界。本文提出的延时模型的准确性依赖于上述估计方法的准确性。

对于清除型 (r,r,k) 队列的期望逗留时间：仅当 $k=1$ 时存在确切的解析解[24,25]；$k>2$ 时仅存在较为粗略的上下界[14,21]。对于非清除 (r,r,k) 队列的期望逗留时间：Fidler 等人[23]给出了较为粗略的非渐进统计型上下界，Wang 等人[16]给出了基于线性变换的确切解析解，并用该解改进了清除型 (r,r,k) 队列的期望逗留时间的上界。

针对 $n \neq r$ 的一般 (n,r,k) 队列的期望逗留时间，Joshi 等人[14]给出了清除型队列的上下界。本文在 Wang 等人[16]工作的基础上第一次给出了非清除 (n,r,k) 队列的确切解析解。

5.2 NWR 数据库延时分析

目前针对 Cassandra 等 NWR 数据库的操作延时分析，主要分为基于基准测试的方法和基于模拟队列的方法。

通过基准测试分析 NWR 数据库性能的工作[3–6]采用主流的分布式数据库基准测试工具 YCSB[26]，分析包括节点数量、副本因子、一致性水平、负载压力等配置变量对读写操作延时的影响。限于基准测试不能穷尽所有的配置组合，这些工作只是各自给出了延时随配置变量变动的粗略趋势。此外，测试结果是基于特定负载下的特定集群得出的，并不能证明其结论具有普遍适用性（完备性）。与之相比，本文给出综合了各配置变量的通用写延时理论模型。基准测试只是用来获取模型的参数、验证模型适用性的辅助手段。

通过模拟队列分析 NWR 数据库性能的主要思路是：1）基于对目标数据库读写操作过程的详细分析建立与之等价的模拟队列网络（Queue Network）；2）通过基准测试确定模拟队列的服务时间、网络传输延时等各种参数；3）针对一些配置变量组合运行模拟队列，以获取目标配置下的延时。如：Huang 等人[7]详细分析了 Cassandra 写操作的执行过程和队列结构；Osman 等人[8]详细分析了 Cassandra 的读操作的队列结构，并使用 Queueing Petri Nets 对其模拟。其他相关工作有[9-13]。相比基准测试，这类工作减轻了测试的运行开销，但由于模拟队列的结构很难完全反映真实集群的内部结构，其参数也很难概括真实集群在各类负载下的实际值，导致基于模拟队列的测试结果的可靠性要差于基准测试。此外其结论同样是粗略且不完备的。

6 结束语

由于 NWR 数据库的广泛应用，对该类数据库的性能分析，尤其是节点数量、副本因子、一致性水平、负载压力对读写操作延时的影响日显重要。现有基于基准测试或模拟队列的分析方法存在结论粗略、完备性欠缺等问题。本文：a）从分析 NWR 数据库写操作的排队模型——非清除 (n,r,k) Fork-Join 队列入手，第一次给出了该类队列的期望逗留时间的解析解，并在独立同指数服务时间分布的模拟 (n,r,k) 队列上验证了解析解的准确性；b）基于上述解析解，第一次给出了 NWR 数据库的理论延时模型。该模型可以反映不同节点数量、副本因子、一致性水平、负载压力、负载均衡策略对写延时的影响。模型的输入为最高一致性（ALL）下的期望延时的基准测试测出值或理论估计值，输出为其他一致性（ONE、QUORUM 等）下的期望延时估计值。相较已有工作，基于该模型得出的结论更为详细和完备。本文在 NWR 数据库 Cassandra 的真实集群上验证了写延时模型的适用性。

未来的研究方向包括：a）借助非清除 (n,r,k) 队列的期望逗留时间的解析解，改进清除型 (n,r,k) 队列的期望逗留时间的上界；b）借助 Limited Processor Sharing (LPS)排队理论[27]，将 Cassandra SEDA 每个阶段的线程池的大小纳入延时模型，从而使其可以反映不同线程数对写延时的影响；c）更为鲁棒的一般服务时间 (i,i,i) 队列的期望逗留时间估计方法，从而进一步减少延时模型的输入参数；d）更为综合的负载（不同读写比例、访问数据的分布、整体数据的分布）的延时模型；e）满足 SLA 延时要求且能够实现集群构建和运行成本最小化的节点数和副本因子的配置组合的计算方法。

参考文献：

[1] Decandia G, Hastorun D, Jampani M, et al. Dynamo [J]. ACM SIGOPS Operating Systems Review, 2007, 41 (6): 205–220.

[2] Lakshman A, Malik P. Cassandra [J]. ACM SIGOPS Operating Systems Review, 2010, 44 (2): 35.

[3] Dede E, Sendir B, Kuzlu P, et al. An evaluation of cassandra for hadoop [C]// Proc of IEEE International Conference on Cloud Computing. 2013: 494–501.

[4] Wang H, Li J, Zhang H, et al. Benchmarking replication and consistency strategies in cloud serving databases: hbase and cassandra [C]// Proc of Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware. 2014: 71–82.

[5] Kuhlenkamp J, Klems M, Röss O. Benchmarking scalability and elasticity of distributed database systems [C]// Proc of the VLDB Endowment. 2014: 1219–1230.

[6] Huang X, Wang J, Yu P S, et al. An experimental study on tuning the consistency of nosql systems [J]. Concurrency and Computation: Practice and Experience, 2017, 29 (12): e4129.

[7] Huang X, Wang J, Bai J, et al. Inherent replica inconsistency in cassandra [C]// Proc of IEEE International Congress on Big Data. 2014: 740–747.

[8] Osman R, Piazzolla P. Modelling replication in nosql datastores [C]// Proc of//International Conference on Quantitative Evaluation of Systems. 2014: 194–209.

[9] Gandini A, Gribaudo M, Knottenbelt W J, et al. Performance evaluation of nosql databases [C]// Proc of European Workshop on Performance Engineering. 2014: 16–29.

[10] Pérez-Miguel C, Mendiburu A, Miguel-Alonso J. Modeling the availability of cassandra [J]. Journal of Parallel and Distributed Computing, 2015, 86: 29–44.

[11] Niemann R. Evaluating the performance and energy consumption of distributed data management systems [C]// Proc of the 10th International Conference on Global Software Engineering Workshops. 2015: 27–34.

[12] Dipietro S, Casale G, Serazzi G. A queueing network model for performance prediction of apache cassandra [C]// Proc of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools. New York: ACM Press, 2017: 186–193.

[13] Huang X, Wang J, Qiao J, et al. Performance and replica consistency

simulation for quorum-based nosql system cassandra [C]// Proc of International Conference on Applications and Theory of Petri Nets and Concurrency. 2017: 78-98.

[14] Joshi G, Soljanin E, Wornell G. Efficient redundancy techniques for latency reduction in cloud systems [J]. ACM Trans on Modeling and Performance Evaluation of Computing Systems, 2017, 2 (2): 12: 1-12: 30.

[15] Wang W, Harchol-Balter M, Jiang H, et al. Asymptotic response time analysis for multi-task parallel jobs [EB//OL]. (2017//10//27) [2017//11//12]. Arxiv: 1710. 00296 [cs. PF].

[16] Wang H, Li J, Shen Z, et al. Approximations and bounds for (n, k) fork-join queues: a linear transformation approach [EB//OL]. (2017//09//08) [2017//09//12]. Arxiv: 1707. 08860 [cs. PF].

[17] Nelson R, Tantawi A N. Approximate analysis of fork//join synchronization in parallel queues [J]. IEEE Trans on Computers, 1988, 37 (6): 739–743.

[18] Welsh M, Culler D, Brewer E. Seda: an architecture for well-conditioned, scalable internet services [J]. ACM SIGOPS Operating Systems Review. 2001, 35 (5): 230–243.

[19] Varma S, Makowski A M. Interpolation approximations for symmetric fork-join queues [J]. Performance Evaluation, 1994, 20 (1–3): 245–265.

[20] Thomasian A, Tantawi A N. Approximate solutions for m//g//1 fork//join synchronization [C]// Proc of Winter Simulation Conference. San Diego: Society for Computer, 1994: 361–368.

[21] Joshi G, Liu Y, Soljanin E. Coding for fast content download [C]// Proc of the 50th Annual Allerton Conference on Communication, Control, and Computing. 2012: 326–333.

[22] Rizk A, Poloczek F, Ciucu F. Computable bounds in fork-join queueing systems [C]// Proc of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. New York: ACM Press, 2015: 335–346.

[23] Fidler M, Jiang Y. Non-asymptotic delay bounds for (k, l) fork-join systems and multi-stage fork-join networks [C]// Proc of the 35th Annual IEEE International Conference on Computer Communications. 2016.

[24] Lee K, Pedarsani R, Ramchandran K. On scheduling redundant requests with cancellation overheads [J]. IEEE//ACM Trans on Networking, 2017, 25 (2): 1279–1290.

[25] Gardner K, Zbarsky S, Doroudi S, et al. Reducing latency via redundant requests: exact analysis [C]// Proc of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. New York: ACM Press, 2015: 347–360.

[26] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with ycsb [C]// Proc of the 1st ACM Symposium on Cloud Computing. New York: ACM Press, 2010: 143–154.

[27] Towsley D, Rommel C G, Stankovic J A. Analysis of fork-join program response times on multiprocessors [J]. IEEE Trans on Parallel and Distributed Systems, 1990, 1 (3): 286–303.